Load Balancing Evaluation

1. Implementation

A total of five load balancing policies have been implemented. They can be divided into two categories: static and dynamic policies.

1.1. Static Policies

Static policies do not take into consideration the state of the system (i.e. the load on the nodes, pending requests per node, response time etc.), but rather try to make a best-effort approximation on how the nodes will be affected by the load. In this category we can find the *Round Robin Policy*, the *Machine Weighted Round Robin Policy* and the *Region Weighted Round Robin Policy*. In the following I will explain how each one of the above mentioned algorithms operates.

1.1.1. Round Robin

Even though this algorithm has a simple operation principle, it is widely used granted its efficiency for distributing work loads among various nodes. Also sometimes referred to as *"next in loop"*, Round Robin fairly distributes the incoming requests by forwarding them to each node one at a time. If there are, for example, three nodes available, denoted *A, B, and C,* the first request will be forwarded to A, the second to B, the third to C, the fourth to A, the cycle continuing until no more requests are to be sent.

1.1.2. Machine Weighted Round Robin

This policy, hence its name, works similarly to the above explained Round Robin, but also takes into consideration the maximum load on the available nodes. A *weight* is set for each node, which in our case represents the maximum concurrent requests allowed for each node. This is useful in the situation when one of your nodes is less able to handle a demanding load than any of the other nodes, thus its assigned weight will be lower. The distribution of the requests is exactly the same as the one presented in the Round Robin algorithm, the only restriction being the weight of the node.

1.1.3. Region Weighted Round Robin

This policy operates similarly to the *Machine Weighted Round Robin* policy, but instead of setting weights to nodes themselves, the appropriate weight is instead set to the regions. Instead of calling the endpoint which sends the request directly to a node, the endpoint which sends the request to a region is instead called. The distribution of the requests among the regions is done one at a time, respecting the weight constraint.

1.2. Dynamic Policies

Dynamic policies change their forwarding strategy based on external factors and results. They offer a better and more realistic approach to a situation where the incoming requests are to be distributed among the available nodes in a more suited way. Before sending a request to a machine, various statistics are taken into consideration, such as latency, response time, number of active requests etc.

1.2.1. Least Response Time

This dynamic policy sends the request to the machine who reported the least response time. Initially the response times are set to 0 and once a request has been finished it updates the node's response time with the value received from the server. Considering that all the requests can be asynchronously sent even before the first request has been completed, a maximum number of concurrent requests has been set, thus limiting the throughput of the forwarding unit.

1.2.1. Least Connections

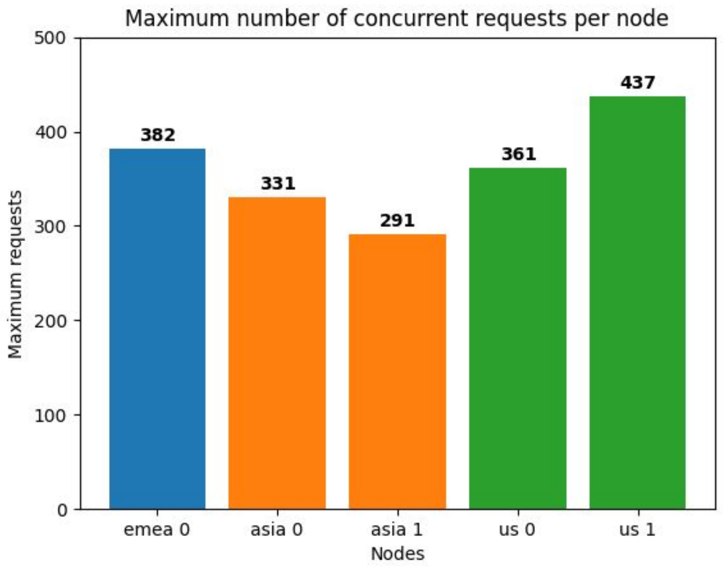
This policy keeps track of the number of active requests (i.e. requests that have been sent, but haven't received a response yet) for each node and sends the next request to the machine that has the least number of active requests. This strategy will favor the nodes that manage to send the response the fastest, thus creating an equitable distribution of requests.

2. Evaluation and results

Limits and characteristics of the system must be first known before trying to implement any of the above mentioned load balancing policies. Several metrics have been collected to better understand the system at hand. After taking into consideration the resulted metrics, the policies have been implemented to best suit the constraints of the system.

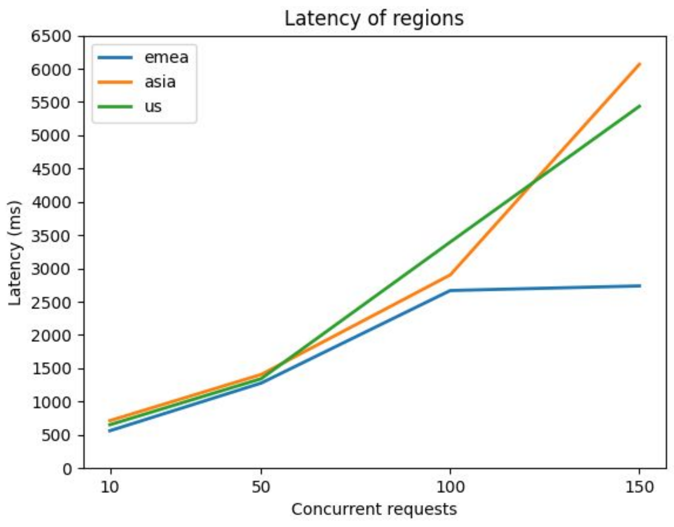
2.1. System Limits Analysis

2.1.1. Maximum number of requests accepted per node



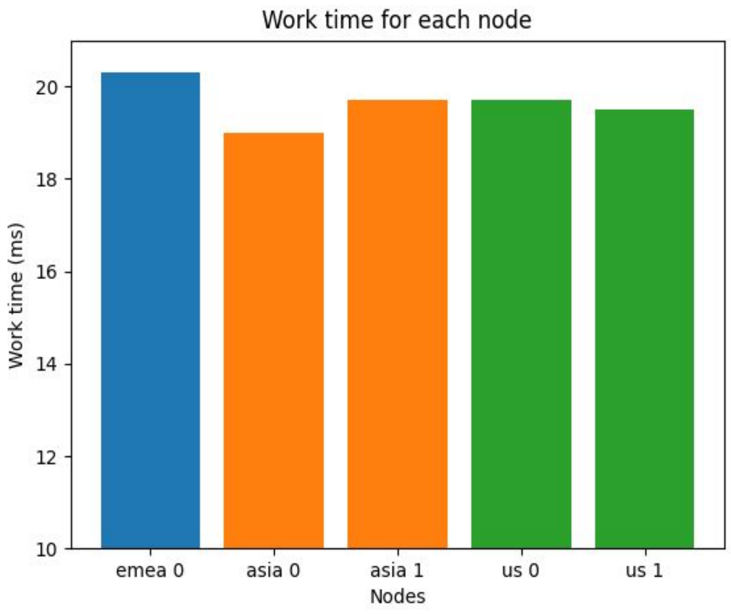
The nodes have been stressed with hundreds of concurrent requests until the services failed and restarted. The figure above shows the results obtained by averaging the numbers from multiple stress tests. As it can be observed, the worst region to perform is *asia*, while the one to handle the most requests is *us*, particularly *us 1*.

2.1.2. Latency of regions



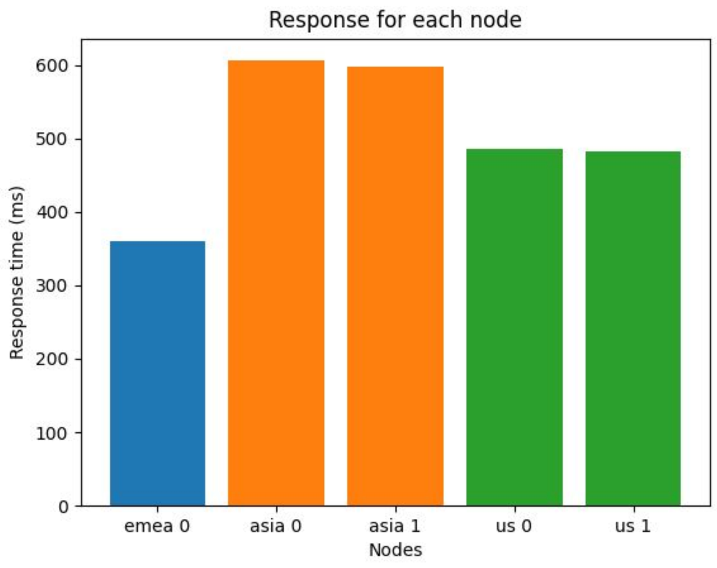
The figure above shows the latency of each region in respect to the load. The lowest latency recorded was for the *emea* region at all times, while *us* and *asia* produce the highest latency when the number of concurrent requests goes past the 100 limit.

2.1.3. Computation times for work requests



Work times have been observed to not differ when the system is on load. Work times are approximately the same for each node. The system was tested by averaging ten requests per node.

2.1.4. Response times without load



Response times show the same difference as the latency analysis. Each node performs the same with the nodes from its region, while globally *emea* has the lowest response time, followed by *us* and *asia*.

2.1.5. System latencies and drawbacks

The current implementation of the forwarding unit produces about 5 - 10 ms of latency for the requests. The majority of the latency is created by limiting the total amount of concurrent requests. This constitutes a drawback but it is necessary for the implementation, as a slow worker is better than a crashed worker.

For the forwarding unit to be the bottleneck of the system, its latency must exceed the response time from a request. For this to be achievable, more than 1000 requests must be sent at a single time to a single node. This way the forwarding unit will limit the throughput, thus creating latency. This issue could be solved by adding more servers or more powerful ones, so the forwarding unit wouldn't limit the number of requests sent.

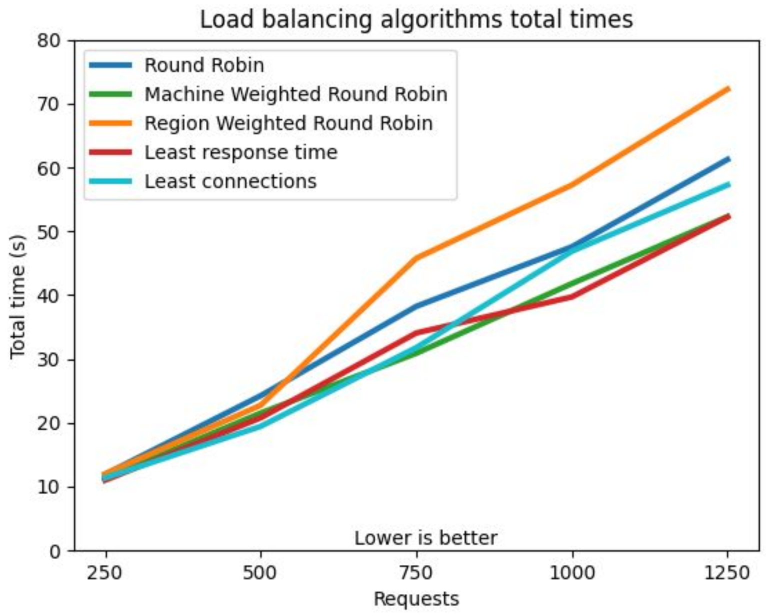
Another part of the system that could create latency is the Heroku service. While no real numbers can be calculated, it is assumed that the latency of the service could be at around 5-10% of the total response time for requests.

The current drawbacks of the systems are the worker's reduced concurrent connections and the latency created by different regions like *asia* and *us*.

2.2. Load Balancing Policies Analysis

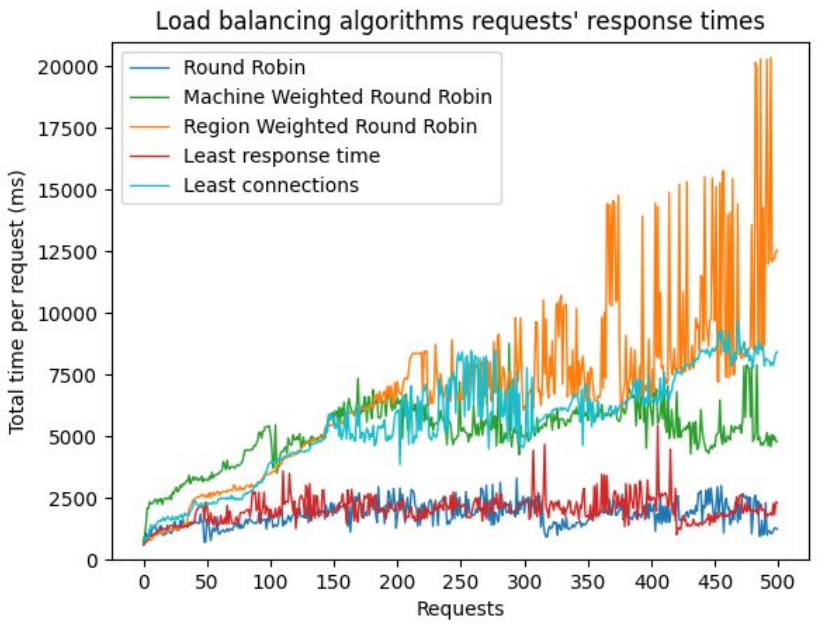
Each of the presented load balancing algorithms have their advantages and disadvantages. Several metrics will be presented in the following to better emphasize each algorithm's properties.

2.2.1. Total time elapsed



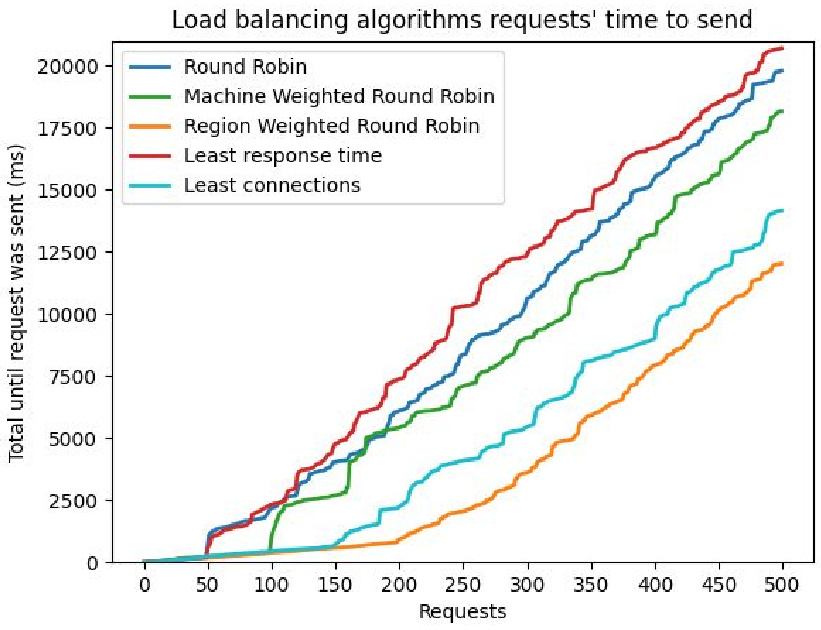
Regarding the total time of processing different number of requests, *Machine Weighted Round Robin* and *Least Response Time* are the best to perform. The latter shows better time to requests ratio as the number of requests increases.

2.2.2. Requests' latency



The figure above presents the response time of each request for an input of 500 requests. The dynamic policies, *Least Response Time* and *Round Robin*, manage to keep the load on the nodes as low as possible, while static policies tend to have an increased load on the machines. *Region Weighted Round Robin* manages to score a 20 second response time for a few requests throughout its execution.

2.2.3. Requests' time to send



The figure shows how much time has elapsed since the first request was sent for each of the 500 probed requests. The straight line at the beginning shows that requests were continuously sent until the maximum concurrent requests limit was hit. Each algorithm implements a different limit to better suit its policy. It can be observed that the *Least connections* and *Region Weighted Round Robin* maintain a lower time-to-send, while *Least response time* focuses on not overloading the nodes, thus maintaining a higher time-to-send.

3. Conclusions

Several conclusions can be drawn from all the measurements and implemented policies. The algorithm to best suit the system at hand was *Least response time* because it did not only maintain a low load on the nodes, but it also managed to achieve the lowest total time overall.

Round Robin also managed to keep nodes on low load, but its high time to send and medium to high total time shows that dynamic policies have an advantage over static policies when not all of the system's constraints are known.

Region Weighted Round Robin performed the worst of all the implemented policies. Even though it maintained the lowest time-to-send, it managed to do that by overloading the nodes, thus increasing their response times and in consequence the total time.